VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROBOTICKÉ RAMENO - NAVIGACE POMOCÍ KAMERY AXIS 214

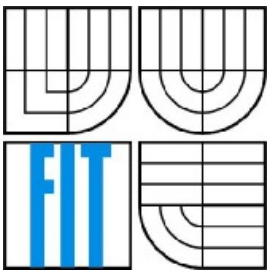
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JOSEF SKLÁDANKA

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROBOTICKÉ RAMENO - NAVIGACE POMOCÍ KAMERY AXIS 214

ROBOTIC ARM - NAVIGATION BY USING AXIS 214 CAMERA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR
Josef Skládanka

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. František V. Zbořil, CSc.

BRNO 2009

Abstrakt

Tato práce se zaměřuje především na rozpoznání pozice kostky, položené na pracovní ploše robotického ramene Mitsubishi Melfa 6 SL, za pomoci digitálního obrazu z kamer Axis 214. Stručně popisuje několik způsobů, jakými lze řešit výpočet pozice objektu v obraze. Detailně pak rozebírá metodu založenou na analýze přerušení linií mřížky vyznačené na pracovní ploše robotického ramene.

Klíčová slova

Robotické rameno, Mitsubishi Melfa 6 SL, IP kamera, Axis 214, zpracování obrazu, pozice objektu v obraze, OpenCV, Python

Abstract

This paper is focused on recognizing the position of a cube placed in the working area of the robotic arm Mitsubishi Melfa 6 SL, using digital image from Axis 214 cameras. It briefly describes few techniques which can be used for detecting the position of an object in the image. The main focus is laid on the implemented method, which detects and analyses gaps in the grid plotted in the robot's working area.

Keywords

Robotic arm, Mitsubishi Melfa 6 SL, IP camera, Axis 214, image processing, position of an object in the image, OpenCV, Python

Citace

Skládanka Josef: *Robotické rameno – navigace pomocí kamery Axis 214.* Brno, 2009, bakalářská práce, FIT VUT v Brně.

Robotické rameno – navigace pomocí kamery Axis 214

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Františka V. Zbořila, CSc. Další informace mi poskytl také Petr Schindler.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Josef Skládanka
20.5.2009

Poděkování

Na tomto místě bych rád poděkoval doc. Ing. Františku V. Zbořilovi, CSc. za odborné vedení, užitečné rady a připomínky. Dále pak Petru Schindlerovi za významnou pomoci při testování. V neposlední řadě rodině a kamarádům za pomoc a podporu při tvorbě této práce.

Obsah

Úvod

Předmětem této bakalářské práce je řešení navigace robotického ramene Mitsubishi Melfa 6 SL za pomoci IP kamer Axis 214. Tato práce je dílčí částí projektu, jehož cílem je vytvořit online laboratoř, která bude sloužit pro podporu výuky umělé inteligence, především řešení problémů z umělého světa kostek. Tyto úlohy mohou studentům usnadnit pochopení základů logického programování a algoritmů sloužících k prohledávání stavového prostoru. Zadání proto směřovalo k záměru navrhnout a implementovat metodu rozpoznávání pozice kostky na pracovní ploše robotického ramene Mitsubishi Melfa 6 SL.

Protože studenti budou laboratoř využívat online, tedy pomocí vzdáleného přístupu, je nutná určitá míra automatizace kontroly dění v laboratoři. Tímto je myšlena navigace pohybu robotického ramene tak, aby byla co nejvíce omezena možnost poškození ramene nebo ostatního vybavení. Protože v laboratoři budou řešeny především problémy ze světa kostek, je nutné získávat aktuální informace o poloze jednotlivých kostek na pracovní ploše. Všechny tyto úkoly budou podpořeny především zpracováním obrazu z kamer. Stejně tak je třeba zajistit vizuální kontrolu dění v laboratoři, tedy přenos obrazu z IP kamer, kterými je laboratoř vybavena, k uživatelům.

Cílem předkládané bakalářské práce je především vytvoření softwaru pro zpracování obrazu – konkrétně výpočet pozice objektu v laboratoři, případně cesty k němu tak, aby robotické rameno neposunulo ostatní kostky nebo nepoškodilo vybavení laboratoře. Potřebnou součástí práce je proto také knihovna umožňující ovládání IP kamer Axis.

První kapitola se zabývá vymezením základních pojmů, které jsou důležité pro další části práce. Ve druhé kapitole je stručně popsána laboratoř, v níž probíhá realizace výše zmíněného projektu, a její vybavení. Třetí kapitola navrhuje způsoby, jakými lze rozpoznávat pozici objektu v obraze a detailně popisuje problematiku vybrané metody, tedy zaměřováni pozice analýzou přerušení mřížky narýsované na pracovní ploše laboratoře. Čtvrtá kapitola pokrývá implementaci a detailnější popis procesu, jakým je kostka zaměřována, a v závěru kapitoly je uvedeno hodnocení výsledků měření a popis problémů, ke kterým v procesu rozpoznávání obrazu dochází. Pátá kapitola stručně popisuje knihovnu pro ovládání IP kamer Axis. Závěr shrnuje dosažené výsledky a předkládá možnosti dalšího pokračování práce na komplexním projektu online laboratoře.

1 Pojmy

V této kapitole jsou vymezeny pojmy používané dále v textu. Jedná se především o termíny související se zpracováním a analýzou digitálního obrazu.

1.1 Digitální obraz

V textu této bakalářské práce je digitálním obrazem myšlena rastrová grafika získaná z IP kamery Axis 214.

Rastrová grafika je způsob popisu (uložení, definice) zpracovávané a zobrazované informace ve formě rastrové matice. Tento popis dat získáme manuálně, syntézou (generováním, nebo převodem z jiného popisu) nebo snímkováním (kamerou atd.). Jeden prvek matice nazýváme pixel.

V rastrové grafice uchováváme informaci (v rastrové matici) pouze o podobě zobrazovaných objektů, nikoli o objektech samotných. [10]

1.2 Barevný model

Barevný model je abstraktní matematický model popisující způsob, jakým mohou být barvy reprezentovány pomocí skupiny čísel, obvykle jako tři nebo čtyři hodnoty jednotlivých barevných složek. Pokud je tento model asociován s přesným popisem, jak mají být jednotlivé komponenty interpretovány, je vzniklá množina barev nazývána barevný prostor [15].

Pro potřeby této práce je důležitý především model HSV odvozený od RGB (viz dále).

1.2.1 RGB

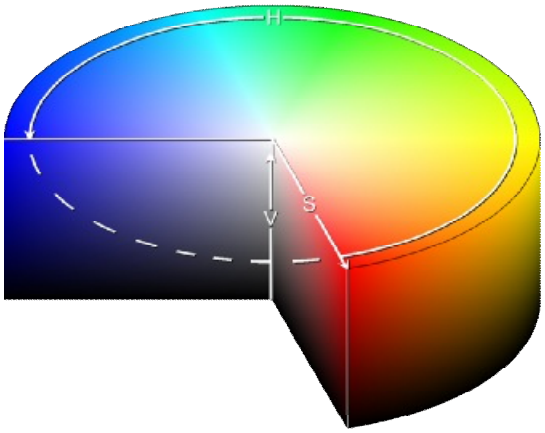
Odpovídá aditivnímu skládání tří základních barev – červené (red), zelené (green) a modré (blue). Výběr těchto tří barev souvisí s fyziologií lidského oka – tyto tři barvy maximalizují rozdíl v odezvě barvocitlivých čípků na sítnici [20].

Hlavním využitím modelu RGB je reprezentace zobrazení barvy v digitálních zařízeních, jako jsou televizory, digitální kamery, scannery a počítače.

1.2.2 HSV

HSV je uživatelsky orientovaný model, který umožňuje intuitivně manipulovat s barvou – vybírat barevný tón (hue), sytost (saturation) a světlost (value) [10]. Proto je obvykle používán v počítačových grafických aplikacích.

Tento model popisuje barvu jako bod uvnitř válce, jehož středová osa sahá od černé (dole) k bílé (nahoře), mezi nimiž jsou odstíny šedi. Úhel kolem osy pak určuje barevný tón, vzdálenost od osy směrem k okraji sytost a vzdálenost na ose koresponduje se světlostí (viz obrázek 1.1) [17].



Obrázek 1.1 – Grafické znázornění modelu HSV (obrázek převzat z [17])

Výše zmíněné vlastnosti jsou však výhodné i během (před)zpracování digitálního obrazu pro účely počítačového vidění. Umožňuje například jednoduchým způsobem z obrazu vyfiltrovat pixely jedné barvy (resp. barevného odstínu) bez ohledu na sytost a světlost. Stejně tak detekce nebarevných (šedých) pixelů je poměrně jednoduchá[1]. Tento fakt je využíván dále v bakalářské práci, jednak při detekci přibližné pozice kostky na pracovní ploše, ale také při přípravě obrazu ke zpracování.

1.3 Histogram

V počítačové grafice rozumíme histogramem statistické měřítko popisující distribuci barev v digitálním obraze. Vodorovná osa grafu reprezentuje odstíny barev, svislá pak počet pixelů spadajících do dané kategorie. Levá strana vodorovné osy představuje černé a tmavé oblasti, střed šedé tóny, a pravá strana pak světlé a bílé pixely (viz obrázek 1.2 B) [18].

Pro účely rozpoznávání obrazu je histogram využíván především k nalezení vhodných mezí pro prahování (viz dále).

1.4 Prahování

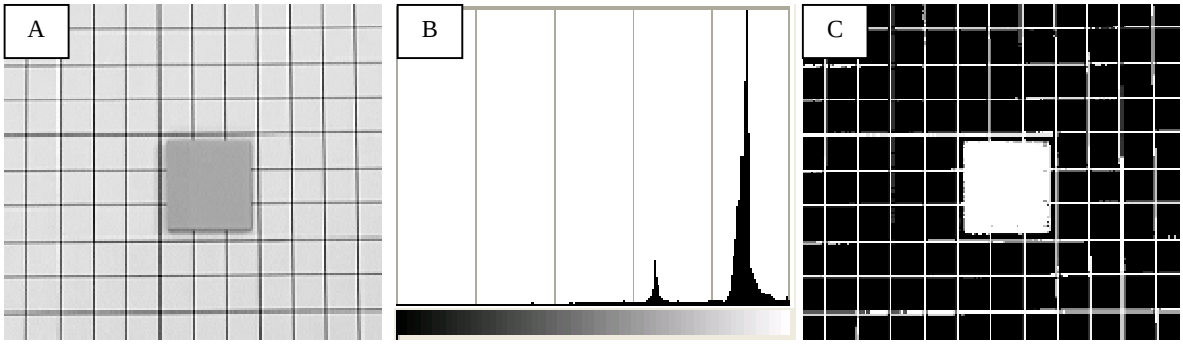
Prvním úkolem při zpracování obrazu je oddělení objektů od pozadí (segmentace obrazu). Prahování (angl. thresholding) poskytuje jednoduchou a vyhovující cestu, jak segmentaci provést na základě rozdílných intenzit barev pozadí a popředí. Stejně tak je prahování použitelné pro oddělení pixelů, jejichž odstín nebo intenzita leží ve specifikovaném intervalu hodnot.

Vstupem pro prahování je obvykle obraz ve stupních šedi. Výstupem je pak v běžných implementacích binární obraz reprezentující segmentaci. Černé pixely odpovídají pozadí a bílé objektům zájmu. Rozhodnutí se provádí na základě jednoho parametru, zvaného práh intenzity. Pixely jsou při jednom průchodu obrazem porovnány se stanovenou hodnotou prahu. Pokud je intenzita barvy větší než práh, je výsledný pixel bílý, v opačném případě černý (případně naopak). [5]

K nalezení správné hodnoty prahu se většinou používá histogram. Obrázek 1.2 B ukazuje příklad bimodálního rozdělení. První lokální maximum v histogramu reprezentuje tmavé pixely

[1] Detekovat šedé pixely není zásadní problém ani v RGB modelu, protože u šedých barev mají všechny tři složky stejnou hodnotu. HSV však dle mého názoru umožňuje jednodušší a přesnější vymezení šedé oblasti.

mřížky a kostky, větší pak světlé pozadí. Výsledek prahování s prahem nastaveným na hodnotu intenzity 170 je zobrazen na obrázku 1.2 C.



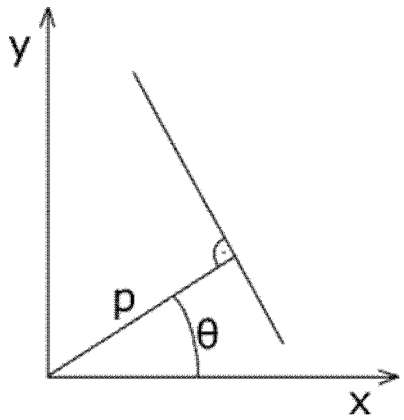
Obrázek 1.2 (A) Zdrojový obrázek, **(B)** Histogram (odpovídající A), **(C)** Výsledek prahování.

1.5 Houghova transformace

Houghova transformace je technika, která je využívána pro oddělení jevů daného tvaru uvnitř obrazu. Protože tento přístup vyžaduje parametrický popis těchto jevů, je klasická Houghova transformace nejčastěji užívána pro detekci běžných křivek, jako jsou přímky, kružnice, elipsy ap. [4]

Houghova transformace používá k nalezení objektů spadajících do dané skupiny tzv. hlasovací přístup. Hlasování probíhá v parametrickém prostoru, ve kterém jsou kandidáti objektů vybráni jako lokální maxima v takzvaném akumulátorovém prostoru, který je vytvořen algoritmem Houghovy transformace.[16]

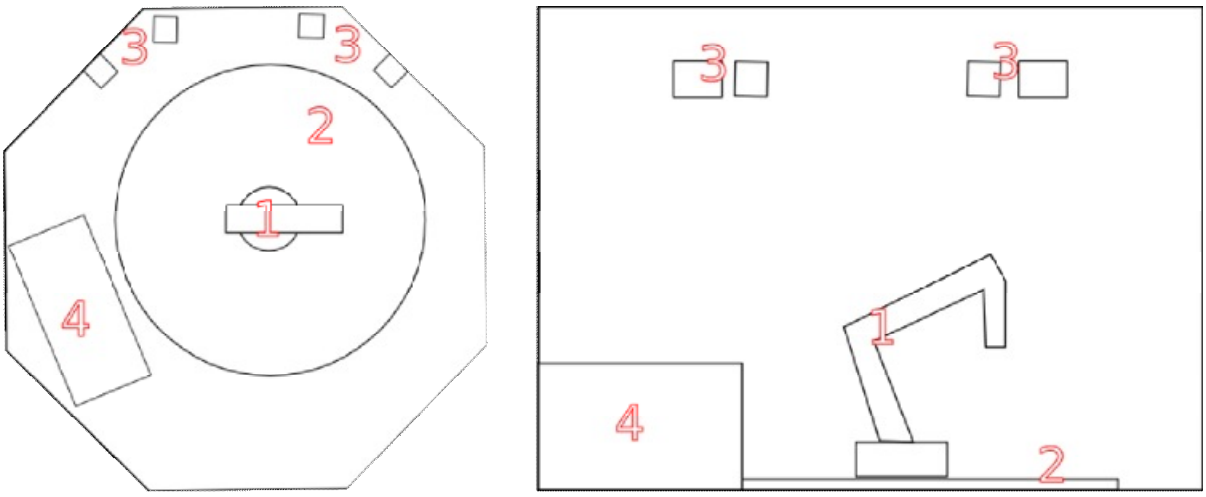
Nejjednodušším případem Houghovy transformace je lineární transformace pro detekci přímek. Tyto lze popsat vztahem $y = kx + q$. Hlavní myšlenka Houghovy transformace spočívá v charakterizování přímky nikoli pomocí souřadnic x a y, ale parametry k (směrnice) a q (úsek vyťatý přímkou na ose y), a tak je každý bod přímky v parametrickém prostoru reprezentován bodem $[q, k]$. Protože tento způsob parametrizace je velmi náročný při výpočtu transformace, předložili Richard Duda a Peter Hart v roce 1972 přístup, který nahrazuje směrnicové vyjádření takzvanou normální formou přímky $p = x \cos(\theta) + y \sin(\theta)$, kde p je vzdálenost přímky od počátku soustavy souřadnic a θ orientace p vzhledem k ose x [3], viz obrázek 1.3.



Obrázek 1.3 Normální forma přímky

2 Laboratoř

V laboratoři se nachází robotické rameno Mitsubishi Melfa 6 SL, řídící jednotka robota, čtyři kamery Axis 214 (připevněné na posuvných lištách zavěšených pod stropem) a počítač (PC), který slouží jako centrální řídící jednotka laboratoře. Přibližné rozmístění objektů v laboratoři ilustruje obrázek 2.1.



Obrázek 2.1 – Schéma laboratoře. 1 – Robotické rameno, 2 – Pracovní plocha, 3 – Kamery, 4 – Stůl s řídící jednotkou a PC

2.1 Robotické rameno Melfa 6 SL

V laboratoři je umístěno robotické rameno firmy Mitsubishi se šesti stupni volnosti, ke kterému je připojen manipulátor od firmy Schmunk. Robotické rameno Melfa 6 SL je zaměřeno především na automatickou, programem řízenou práci v průmyslových aplikacích. Konkrétní způsob užití samozřejmě závisí především na typu připojeného nástroje. V tomto případě je připojena mechanická ruka se dvěma prsty. S využitím vyměnitelných nástavců je možné ruku uzpůsobit k uchopení různých typů předmětů [13].

Robotické rameno je standardně ovládáno programem v řídící jednotce, která poskytuje různá rozhraní pro komunikaci s uživatelem. Pro naše účely je důležitý především COM port, pomocí kterého může řídící jednotka komunikovat s počítačem. Tím je umožněno především on-line ovládání robota bez nutnosti předem programovat řídící jednotku. Ovládáním robotického ramene a vytvořením knihovny pro programovací jazyk Python, pomocí které lze s robotem komunikovat, se ve své bakalářské práci [13] zabývá Petr Schindler.

2.2 Kamery Axis 214

Axis 214 je profesionální IP kamera určená především pro zabezpečení a vzdálený dohled. Streamovaný obraz z kamery může být po síti v reálném čase přenášen ve formátu MPEG-4, nebo jako Motion JPEG. Tento stream bude poskytován uživatelům laboratoře pro vizuální kontrolu.

Pro zpracování obrazu je však vhodnější použít data, která neprošla ztrátovou kompresí. Zde je využita další vlastnost IP kamery – možnost získání samostatného snímku ve formátu BMP. Nekomprimovaná bitmapa je sice náročnější na online přenos, avšak krátká prodleva při získání snímku je vyvážena kvalitou dat pro další zpracování. Prodleva je taktéž výrazně minimalizována propojením kamer a řídícího počítače 1Gbit LAN sítí.

2.2.1 Technická specifikace kamery AXIS 214

Senzor	1/4" HAD CCD
Ohnisková vzdálenost	4.1 - 73.8 mm
Světelnost	F1.3 - 3.0
Závěrka	1/10000 - 1 s
Zoom	18x optický, 12x digitální
Komprese videa	MPEG-4 Part 2, Motion JPEG
Maximální rozlišení	704x576 pixelů
API	VAPIX®

Kamery poskytují funkce pro noční vidění, automatické nastavení clony a ostření, detekci pohybu v obraze, automatické vytváření a archivaci záznamů atd. Pro účely rozpoznávání obrazu je však důležitá především kvalita objektivu, která je na vysoké úrovni, a dále maximální rozlišení obrazu, které je bohužel poměrně malé, a proto je nutné prostor laboratoře při zpracování segmentovat na menší části, které se zpracovávají samostatně.

3 Návrh řešení

V této kapitole jsou rámcově probrány dvě metody, kterými lze rozpoznávat pozici objektu v obraze (triangulace, stereogrammetrie), a detailněji je popsána problematika implementované metody, tj. zaměřováni pozice analýzou přerušení v mřížce narýsované na pracovní ploše laboratoře.

3.1 Měření vzdálenosti v obraze

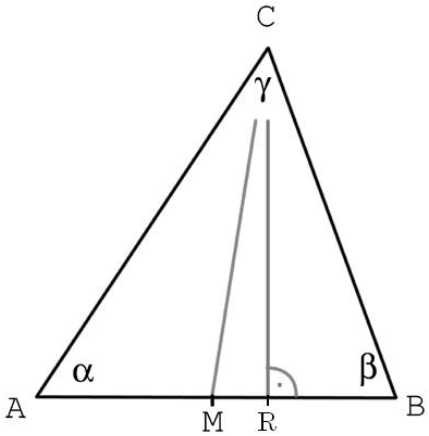
Pro úspěšnou navigaci robotického ramene v laboratoři je důležité znát umístění objektů, jejichž poloha se nemění – kamery, stůl s kontrolní jednotkou – a především pozice kostek, se kterými bude robot pracovat. Při řešení úloh ze světa kostek je nezbytně nutné rozpoznat pouze počáteční konfiguraci „světa“ a na jejím základě inicializovat stavovou databázi. Laboratoř však není součástí dokonalého umělého světa, a proto je pomocí rozpoznávání obrazu nutné průběžně testovat, zda při přesunu kostky nedojde ke kolizi s okolím – tedy na základě analýzy vizuálních dat najít bezpečnou trasu, po které se robotické rameno bude pohybovat. Stejně tak, pokud se například při pokusu o postavení věže kostky rozsypou, je možné pomocí kamer zaměřit pozice jednotlivých dílů a přesunout je do výchozí polohy.

Další text se zabývá především rozpoznáváním polohy kostky na pracovní ploše robota. Zmiňuje způsoby, jak lze kostku zaměřit – optickou triangulaci a fotogrammetrii – a následně implementovanou metodu, založenou na analýze „děr“ v předkreslené mřížce, na které jsou kostky umístěny.

3.1.1 Triangulace

Triangulace je způsob získávání souřadnic a vzdáleností, který využívá elementární geometrii. Provádí se trigonometrickým výpočtem. První krok spočívá v sestrojení pomyslného trojúhelníku, u kterého je známá délka jedné strany zadané referenčními body a velikost úhlů v referenčních bodech. Jednoduchým výpočtem pak lze zjistit souřadnice zaměřovaného bodu (viz obrázek 3.1) [21].

Pokud by laboratoř nebyla vybavena speciálním zařízením, pozice kostek zaměřená touto metodou by pravděpodobně byla zatížena značnou chybou. Pro úspěšné zaměření objektu je nutné znát úhly svírané stranami trojúhelníka v referenčních bodech s co největší přesností. Proto je třeba obě kamery nastavit tak, aby těžiště zaměřované kostky leželo na přímce procházející optickým středem snímku. Úhel natočení kamery se však mění skokově, nikoli kontinuálně, a proto by pro každý snímek bylo třeba vypočítat korekce. Stejně tak nelze spoléhat na to, že čidla otočení kamery poskytují přesné údaje – nejsou na to primárně určena. Časem by tedy pravděpodobně začalo docházet k chybám způsobeným načítáním jednotlivých odchylek.

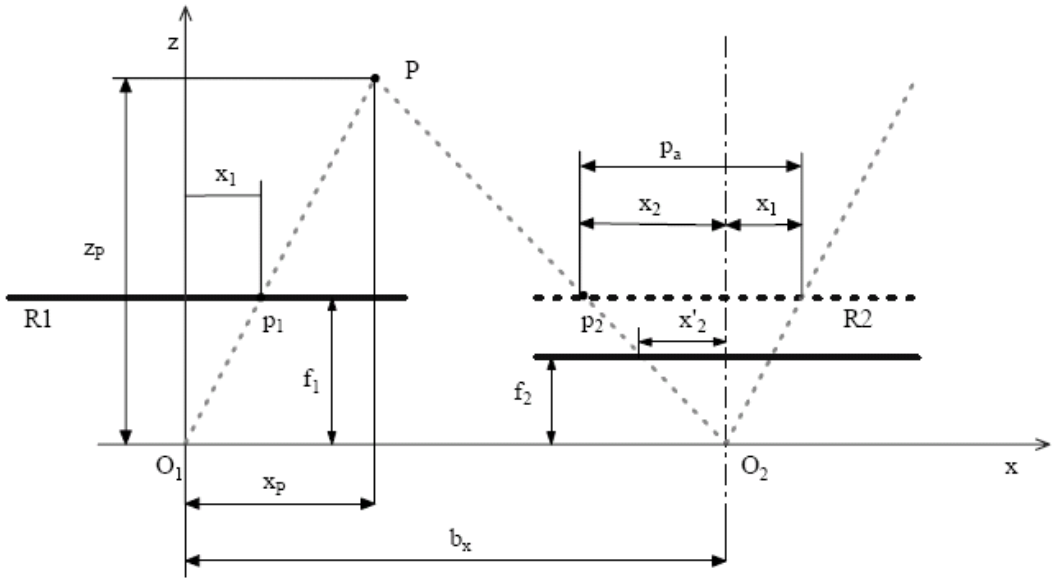


Obrázek 3.1 – Pozorovatel v bodě A změří úhel α a pozorovatel v bodě B úhel β. Pokud známe souřadnice bodů A a B, je možné užitím sinové věty vypočíst souřadnice bodu C a vzdálenost RC. (převzato z [21])

3.1.2 Stereofotogrammetrie

Stereofotogrammetrie je metoda založená na porovnání dvou snímků téže scény. Výpočty probíhají nad dvěma snímky z kamer posunutých pouze po jedné ose. Obecně však lze počítat i se snímky z kamer vůči sobě téměř libovolně posunutých a otočených (samozřejmě pouze pokud jejich polohu známe) užitím transformace souřadnic, a tím problém opět převést na posunutí pouze po ose X.

Pro výpočet prostorových souřadnic je třeba ve snímcích najít tzv. korespondující body – zobrazení téhož reálného bodu na jednotlivých fotografiích. Pro tyto body lze, s využitím údajů o optických vlastnostech objektivu, vypočítat příslušné „světelné paprsky“, které se protnou na souřadnicích zkoumaného bodu (viz obrázek 3.2). Proces výpočtu souřadnic popisuje například [14] nebo [22].



Obrázek 3.2 – Princip stereofotogrammetrie pro kamery s paralelními optickými osami. $O_{1,2}$ – optické středy kamer, $f_{1,2}$ – ohnisková vzdálenost, P – zaměřovaný bod, $p_{1,2}$ – projekce bodu P do plochy snímané kamerou. (převzato z [14])

Nejprve jsem se pokusil pro zaměření pozice kostky použít stereofotogrammetrii, především proto, že poskytuje informace nejen o 2D pozici kostky na pracovní ploše, ale navíc lze z dat disparity odhadnout vzdálenost od kamery – tedy polohu ve třetím rozměru. Pro jednodušší experimentování s tímto postupem jsem se rozhodl použít OpenCV dle kapitol 11 a 12 v [2].

Nejprve jsem experimentoval se zjištěním kalibračních údajů kamery s použitím OpenCV verze 1.0. Kalibrace probíhá v několika krocích – nejprve se vytvoří několik snímků scény, ve které se nachází šachovnice. OpenCV dokáže detekovat rohy polí šachovnice pomocí funkce `cvFindChessboardCorners()`. Množina souřadnic z jednotlivých obrázků je pak předána na vstup funkci `cvCalibrateCamera2()`, která vypočte kalibrační matice a hodnoty koeficientů pro redukci zakřivení a dalších vad způsobených optikou kamery. Z těchto údajů dále funkce `cvInitUndisortMap()` vypočte parametry potřebné pro odstranění zkreslení (angl. undistortion), jež provede metoda `cvRemap()`.

Protože kalibrace obrazu vykazovala kvalitní výsledky, pokračoval jsem dále ke stereovizi. Metody pro zpracování stereo obrazu, jsou však, jak jsem zjistil, implementovány až ve verzi 1.1.pre1a, přestože v [2] je explicitně zmíněna verze 1.0. S přechodem na novou verzi se však začala kalibrační funkce chovat jinak, než ve verzi 1.0. Výsledné kalibrační matice vykazovaly značné rozdíly a výsledky aplikace těchto matic byly na první pohled chybné. Experimentálním způsobem jsem zjistil, že funkce `cvCalibrateCamera2()` z verze 1.1 vyžaduje, aby plocha zabraná na obrázku šachovnicí byla vždy co největší. Jen v takových případech poskytovala použitelné výsledky.

Protože pro stereokalibraci se používá téměř identický přístup jako při výše popsané kalibraci samotné kamery – obsahuje pouze další kroky spočívající v rozpoznání identických bodů v obraze z obou kamer – rozhodl jsem se od použití stereovize upustit. Pro stereokalibraci v prostředí robotické laboratoře není možné zajistit dostatečné množství snímků šachovnice pokrývajících většinu plochy obrazu na obou kamerách současně. Stejně tak, dle mého názoru, není současná implementace stereovize příliš spolehlivá – především proto, že funkce, které se pro stereokalibraci používají, produkuji v každé verzi knihovny naprosto rozdílné výsledky.

3.1.3 Analýza přerušení v mřížce

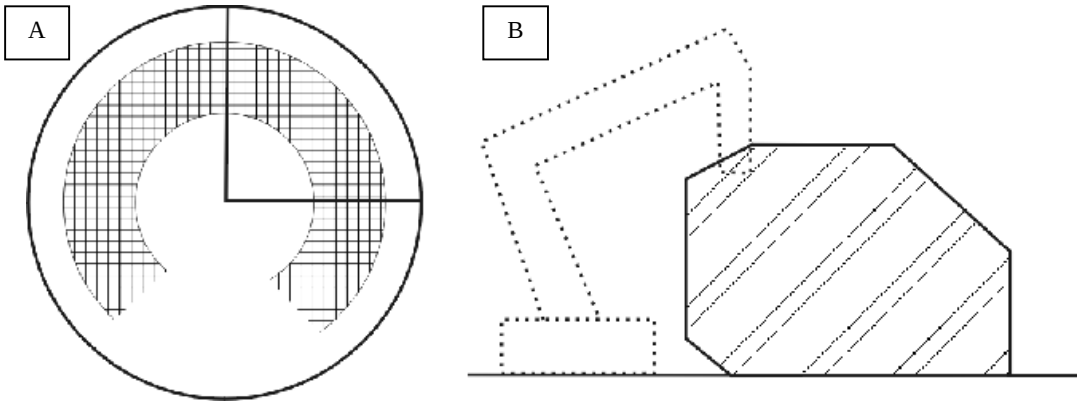
Protože pokus o využití stereovize nevedl ke kvalitním výsledkům, bylo nutné nalézt jiný postup, jak pozici kostky na pracovní ploše pokud možno přesně zaměřit pouze s využitím digitálního obrazu z IP kamer Axis 214. Protože tyto kamery mají velmi malé maximální rozlišení (viz kap. 2.2.1), není možné získat kvalitní informace o pozici zaměřovaného objektu pouze z celkového pohledu na pracovní plochu. Je nutné ji nějakým způsobem rozdělit na menší celky, které lze zpracovat samostatně.

Můj první nápad spočíval ve vytvoření sítě bodů s přesně zaměřenými pozicemi, které by v obraze plnily funkci zaměřovacích křížků, podle kterých je možné obraz z kamery rektifikovat (tedy upravit pomocí posloupnosti lineárních transformací otočení, změny velikosti a změny úhlu) tak, aby výsledný obraz co nejpřesněji zachovával úhly. Pro takovýto obraz lze jednoduše vypočítat, jaké vzdálenosti v realitě odpovídá šířka a délka pixelu.

Tento postup však nebyl příliš spolehlivý, především proto, že kvůli nízkému rozlišení kamery nebyl často detekován dostatečný počet kalibračních bodů na to, aby bylo možné přesně určit, v jaké části pracovní plochy se zaměřovaný objekt nachází, a tedy vybrat patřičnou sadu parametrů pro transformace. Proto jsem se rozhodl nahradit síť jednotlivých bodů mřížkou.

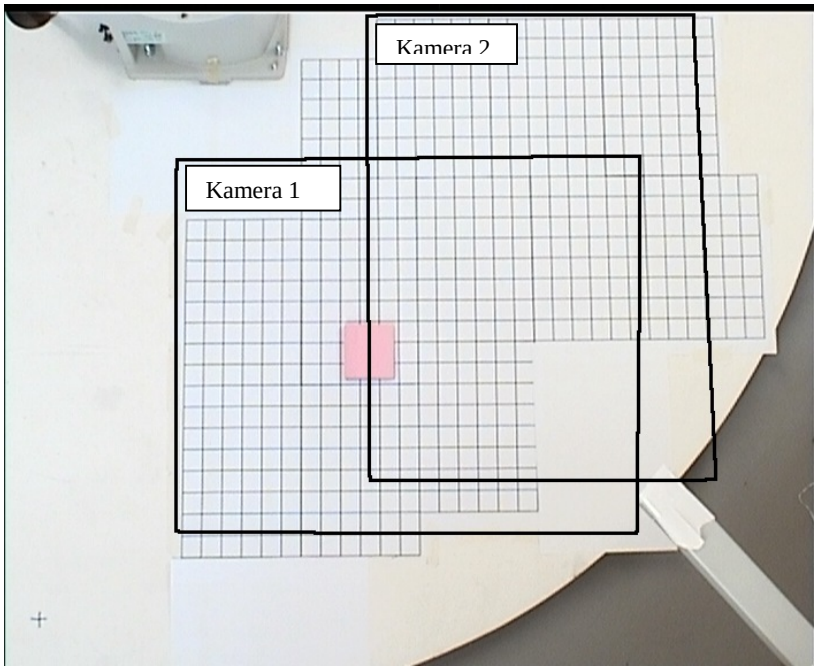
Prvním krokem bylo vyměření prostoru na pracovní ploše, ve kterém se robotické rameno může efektivně pohybovat. Tato plocha je značně omezena především faktem, že při finální aplikaci

bude rameno zvedat kostky z různých výškových hladin, a je tedy nutné, aby mohlo danou kostku zvednout nahoru, ale samozřejmě i položit na pracovní plochu. Další omezení představuje fakt, že kvůli kabelům vedoucím k manipulátoru a kamerám umístěným za posledním kloubem ruky není možné s ramenem pohybovat libovolně, ale je třeba neustále hlídat orientaci posledních dvou kloubů, aby se kabely nezamotaly. Přibližnou představu o prostoru pracovní plochy a efektivní pracovní výšce lze získat z obrázku 3.3.



Obrázek 3.3 – **(A)** Půdorys pracovní plochy, **(B)** Průřez efektivním pracovním prostorem

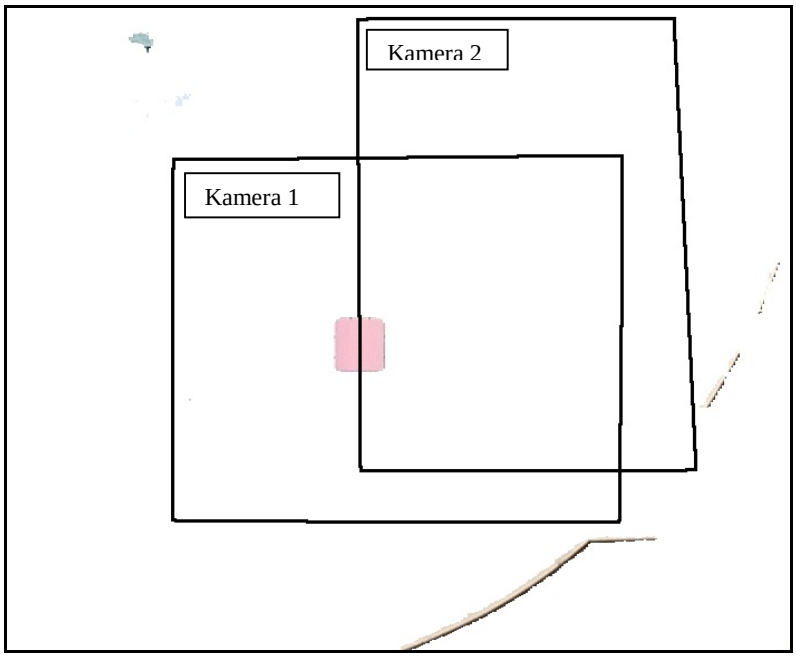
Stále je však potřeba nějakým způsobem vyřešit problém způsobený nedostatečným rozlišením IP kamer, které jsou v laboratoři nainstalovány. Proto detekce a zaměření kostky probíhá v několika krocích. Prvním z nich je získání obrazu celého segmentu pracovní plochy. Přestože tento obraz není dostatečně kvalitní na to, aby v něm bylo přímo možné kostku zaměřit, lze jej využít alespoň k přibližnému určení polohy na pracovní ploše. Tento krok je důležitý proto, že kamery jsou nastaveny tak, aby při přiblížení na takovou vzdálenost, kdy je možné poměrně přesně vyměřit souřadnice kostky, pokrývaly každá jinou část pracovní plochy. Tyto myšlené segmenty se samozřejmě překrývají tak, aby byla potlačena možnost toho, že některá z kostek bude položena přesně na jejich hranici, a v důsledku toho by nebyla celá viditelná na žádném z přiblížených obrazů. Pro první kvadrant pracovního prostoru je rozdělení znázorněno na obrázku 3.4.



Obrázek 3.4 – Plocha pokrytá kamerami při přiblížení.

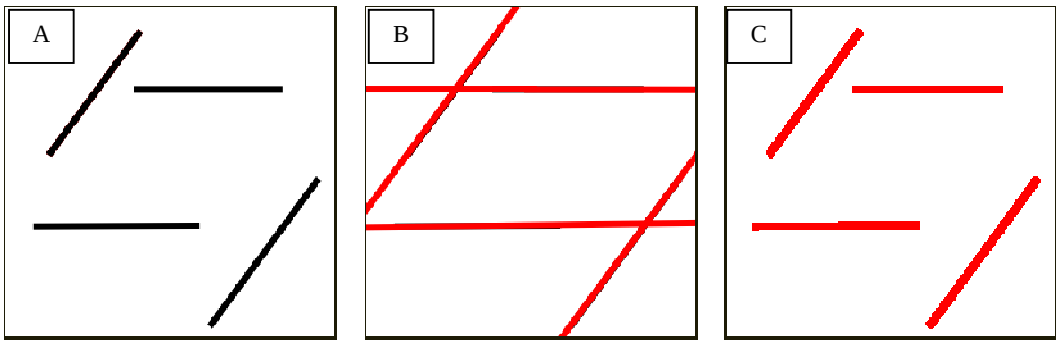
Dalším krokem je tedy rozhodnutí, kterou kamerou bude kostka zaměřována. Tento proces lze pro názornost rozdělit do několika dílčích kroků, i když jsou ve skutečnosti prováděny současně. Nejprve jsou z obrazu vyfiltrovány všechny nebarevné pixely, čímž je efektivně odstraněna mřížka a zůstanou jen barevné plochy reprezentující jednotlivé kostky. Na tento redukovaný obraz jsou promítnuty obdélníky reprezentující záběry kamer a v každém z nich je vypočteno množství barevných pixelů.

Pro další zpracování se použije ta kamera, v jejímž záběru bylo detekováno více barevných pixelů. Je-li v obou obdélnících detekován stejný počet pixelů, pak se (podle nastaveného algoritmu) pro zaměřování použije kamera 2.



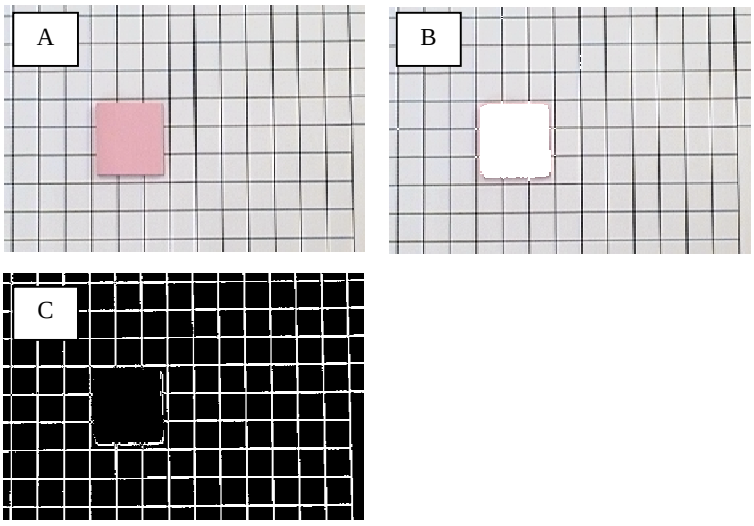
Obrázek 3.5 – Výběr kamery pro další zpracování. V tomto případě je použita kamera číslo jedna, protože se v jejím záběru nachází celá kostka.

Nyní víme, kterou kameru použít pro měření, a tak použijeme přiblížený záběr kamery k zaměření kostky. Protože zaměření prakticky probíhá rozpoznáváním děr ve mřížce, které jsou způsobeny kostkou na ní položenou, je nutné rozpoznat jednotlivé linie tvořící mřížku. K tomu lze využit například Houghovu lineární transformaci. V knihovně OpenCV je dostupná jak standardní implementace Houghovy lineární transformace (dále SHT), tak progresivní pravděpodobnostní Houghova transformace (angl. Progressive Probabilistic Hough Transform, dále PPHT). Mezi těmito implementacemi je na první pohled patrné především to, že PPHT oproti SHT rozpoznává i délku čáry (viz obrázek 3.6). Podle [11] je také PPHT méně náročná na výpočetní čas, protože pro detekci linií nepotřebuje projít každý pixel zkoumaného obrazu. (Pro detailnější popis výkonnostních a kvalitativních rozdílů mezi SHT a PPTH doporučuji zmíněnou publikaci.)



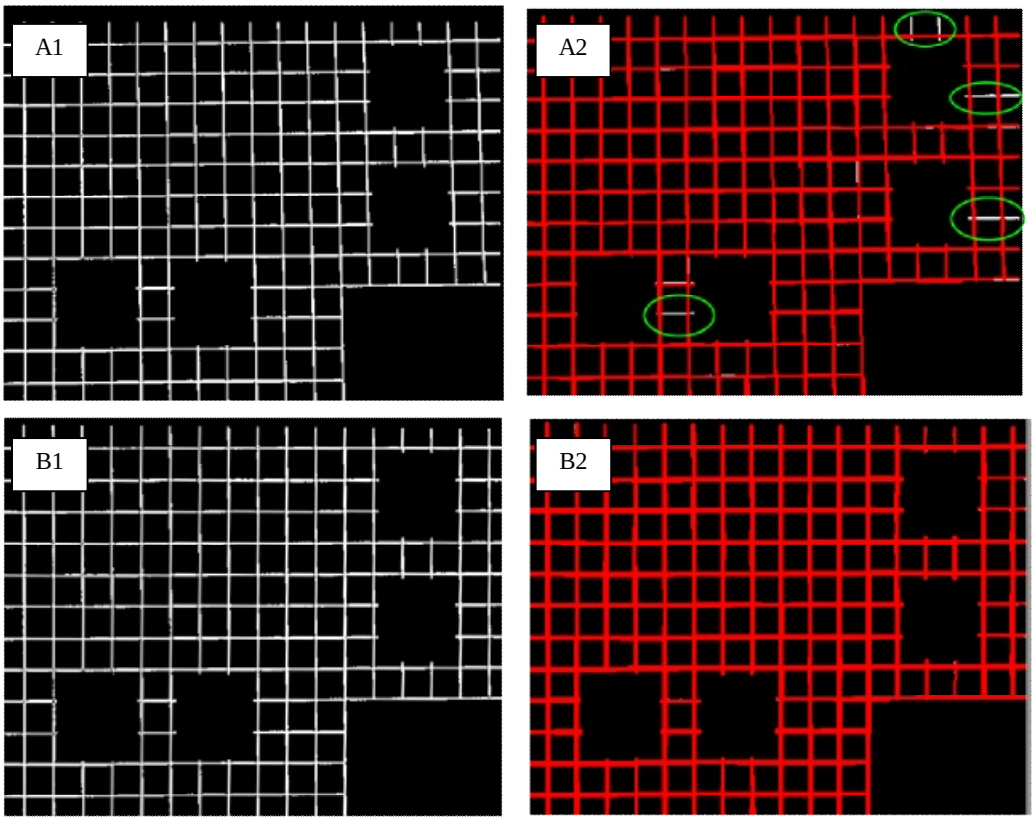
Obrázek 3.6 – Rozdíly mezi výsledky různých algoritmů pro Houghovu transformaci. **(A)** Vstupní data, **(B)** Standardní Houghova transformace, **(C)** Progresivní pravděpodobnostní Houghova transformace.

Detekci mřížky pomocí PPHT samozřejmě předchází několik přípravných kroků, protože Houghova transformace předpokládá na vstupu binární obraz. Pro redukci barevného prostoru je zvolena metoda prahování (viz kap. 1.4). Protože při prahování rozhoduje pouze hodnota intenzity pixelu a barva nehraje roli, jsou nejprve ze zkoumaného obrazu odstraněny všechny barevné pixely – tedy kostky. Pak následuje převod obrazu na stupně šedi a prahování. Jednotlivé kroky jsou znázorněny na obrázku 3.7.



Obrázek 3.7 – Příprava obrazu před detekcí čar. **(A)** Vstupní obraz, **(B)** Obraz po odstranění barevných pixelů, **(C)** Výsledek prahování.

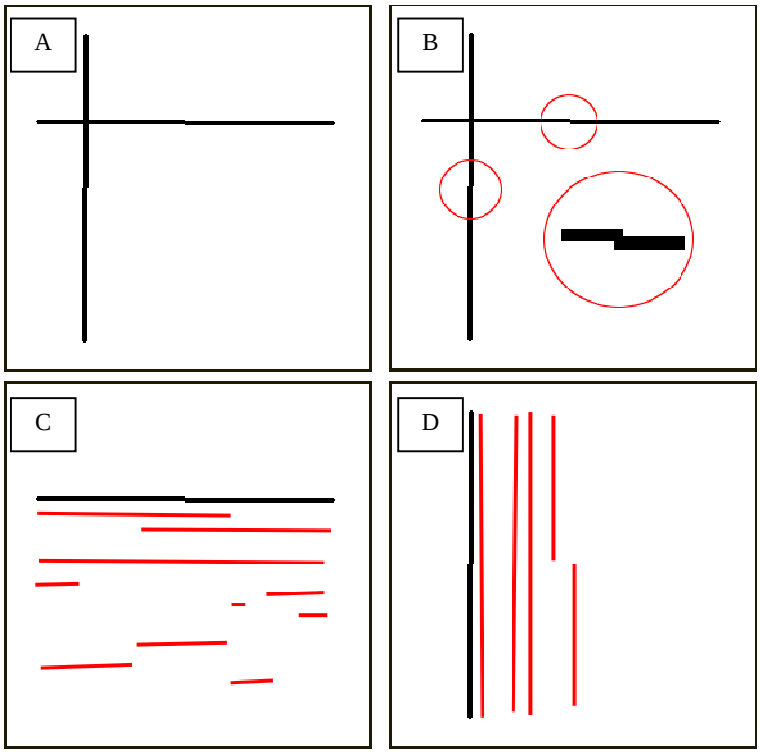
Před použitím Houghovy transformace je také vhodné obraz rektifikovat. Výrazné rozdíly ve kvalitě výsledků, alespoň u implementace PPHT, lze pozorovat především při detekci krátkých čar, například mezi vedle sebe položenými kostkami. Ačkoli je obraz rektifikací změněn jen minimálně, jsou rozdíly ve výsledku Houghovy transformace často výrazné. Patrné jsou na obrázku 3.8.



Obrázek 3.8 – Rozdíly ve výsledku Houghovy lineární transformace. **(A1, A2)** Nerektifikovaný obraz, **(B1, B2)** Rektifikovaný obraz

Nyní tedy jsou jednotlivé linie mřížky detekovány. Z principu PPHT však vyplývá, že čáry nejsou rozpoznávány jako jednolitý celek, ale spíše jako množství na sebe navazujících a překrývajících se fragmentů (viz obrázek 3.9). Například na obrázku 3.8 B detekuje PPHT 270 čar. Spojitých linií je však na obrázku pouze 45.

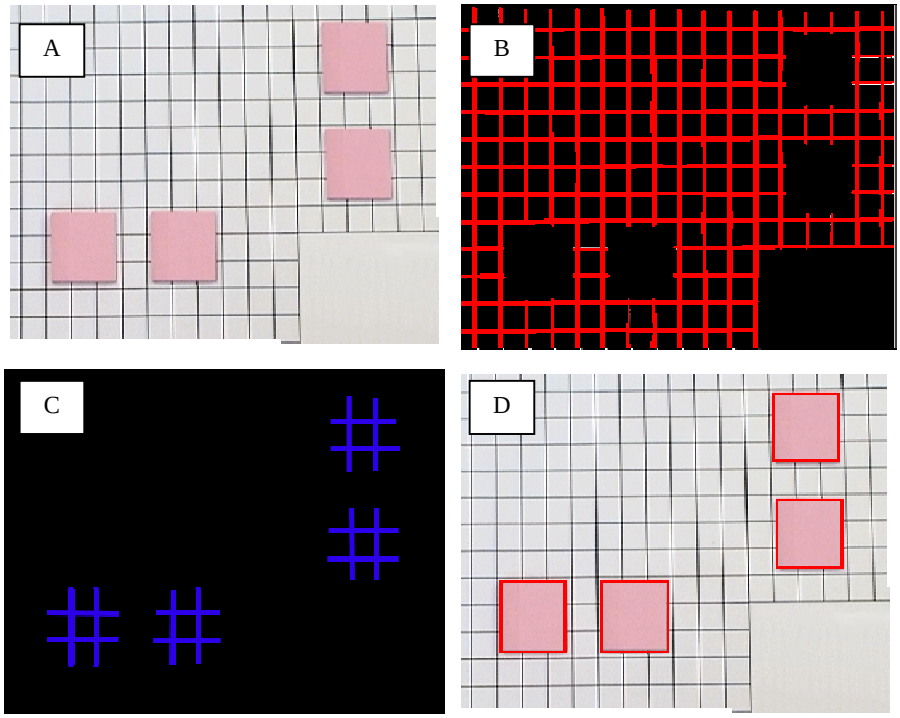
PPTH totiž nevyšetřuje veškeré body v obraze, ale je založeno na přístupu, který by se dal zařadit mezi Monte Carlo metody. Houghova transformace detekuje kolineární body jako lokální maxima ve speciálním parametrickém prostoru. PPHT minimalizuje množství potřebných výpočtů tím, že využívá rozdíly v podílu potřebných hlasů, uvnitř parametrického prostoru, které jsou potřeba pro spolehlivou detekci při různém počtu podpůrných bodů této čáry. Počet zkoumaných bodů však není třeba specifikovat v průběhu (angl. ad-hoc) nebo před započetím výpočtu, jako je tomu u pravděpodobnostní Houghovy transformace (angl. Probabilistic Hough Transform), ale je to funkce vlastní složitosti dat [11]. Nejvýznamnější důvod fragmentace je dle mého názoru fakt, že čáry v obraze jsou širší než jeden pixel. Proto je každá čára detekována několikrát. Tento jev by se nejspíše dal odstranit například zúžením čar (angl. thinning) jak je popsáno v [3].



Obrázek 3.9 – Fragmenty při detekci čar pomocí PPHT. **(A)** Vstupní obrázek, **(B)** Úsečky na obrázku nejsou zcela rovné, taktéž jsou širší než 1px, proto budou detekovány ve více fragmentech. **(C)** a **(D)** Znázornění všech částí, které na vstupních černých úsečkách detekovala PPHT. Dohromady je fragmentů patnáct.

První krok proto spočívá ve sjednocení fragmentovaných linií mřížky. Tento proces samozřejmě probíhá zvlášť pro vodorovné a svislé čáry. Výsledkem jsou prostorově seřazené seznamy horizontálních a vertikálních úseček. Ty však nejsou skutečným předmětem zájmu. Protože chceme zaměřit polohu kostky na této mřížce položené, je nutné získat informace o tom, kde je mřížka přerušena. Tato operace by mohla být volně popsána jako nalezení doplňku k množině úseček na jednotlivých řádcích.

Nakonec už je potřeba jen pole „děr“ převést na lepší reprezentaci prostoru, který na obrázku vymezují. Pro potřeby zaměřování pozice kostky jsou důležité dva údaje – mezi kterými čarami mřížky je kostka položena, a také poloha a rozměry obdélníku vytvořeného na obraze položenou kostkou. Protože je obraz rektifikován, lze oba údaje použít k vypočtení polohy.



Obrázek 3.10 – Detekce kostek. **(A)** Vstupní obraz, **(B)** Mřížka detekovaná Houghovou lineární transformací, **(C)** Vypočtené mezery ve mřížce, **(D)** Rozpoznané kostky.

4 Implementace

Tato kapitola se zabývá vlastní implementací. Obsahuje stručný popis použitých nástrojů následovaný popisem postupu získání obrazu z kamery. Dále rozebírá způsob předzpracování obrazu a rozpoznání pozice kostky.

4.1 Použité nástroje

4.1.1 Python

Python je dynamický objektově orientovaný programovací/skriptovací jazyk. Je vhodný především pro rychlé prototypování v mnoha oblastech vývoje softwaru. Nabízí silnou podporu integrace s dalšími nástroji a jazyky jako jsou .NET (IronPython), CORBA, Java (Jython – implementace Pythonu běžící pod Java Virtual Machine). Pro Python je možné naprogramovat výpočetně náročné části kódu jako moduly v jazycích C/C++ nebo pomocí tzv. wrapperů používat kód v C/C++, který původně nebyl jako modul pro Python zamýšlen.

Protože Python je interpretovaný jazyk, není, podobně jako Java, Ruby či C#, kompilován přímo do strojového kódu, ale pouze do mezikódu, který je prováděn interpretem. Tento přístup sice může způsobovat pomalejší vykonávání programu, ale současně poskytuje vysokou míru přenositelnosti. V současné době je podporován celou řadou velkých platforem, jako jsou Windows, Linux/Unix, OS/2, Mac, Amiga a další. Protože je interpret Pythonu naprogramován v jazyce C, je možné jej s minimálním úsilím přenést do prostředí, pro které je k dispozici kompilátor jazyka C. Jedním z těchto pokusů je například intepret Pythonu pro PocketPC.

Další význačnou kvalitou Pythonu je široká škála knihoven ve standardní distribuci, které pokrývají oblasti od podpory asynchronních procesů, přes sítové protokoly, až po modul pro práci se zip archivy. Spousta dalších knihoven je volně přístupná na Internetu. Python je vyvíjen jako open-source produkt pod Python licence, která dává možnost jej bez omezení používat a upravovat dokonce i pro komerční užití [12].

Z hlediska klasifikace jazyků není Python čistě objektový, protože podporuje více přístupů k programování (např. funkcionální), ale také proto, že standardní knihovna nemá z historických důvodů jednotné, plně objektové rozhraní. Tento fakt však Pythonu nijak neubírá na použitelnosti.

Jednou z jeho dalších předností je i způsob zápisu zdrojového kódu, který definuje blok jako řádky se stejnou úrovní odsazení. Programátor je tak nucen produkovat poměrně přehledný a čitelný kód. Samozřejmostí je i podpora generování dokumentace ve formátu html, stejně jako je tomu například u Javy, a hierarchické členění nejen do knihoven, ale i do balíků tvořených adresářovou strukturou.

4.1.2 OpenCV

OpenCV je knihovna pro podporu počítačového vidění, jejíž vývoj započala společnost Intel v roce 1999. Je volně použitelná pro výzkumné a komerční účely v rámci BSD licence. Tato knihovna je multiplatformní a lze ji využívat ve Windows, Linuxu, Max OS X, PlayStation a mnoha vestavěných zařízeních. Zaměřuje se především na zpracování obrazu v reálném čase a dokáže využít

optimalizační knihovny IPP (Integrated Performace Primitives) vytvořené taktéž firmou Intel. Verze knihovny OpenCV optimalizovaná pro procesory Cell dosahuje dle [6] na jednom PlayStation 3 výkonu srovnatelného s 27 procesory Intel Core2Duo 2.4 GHz. [19]

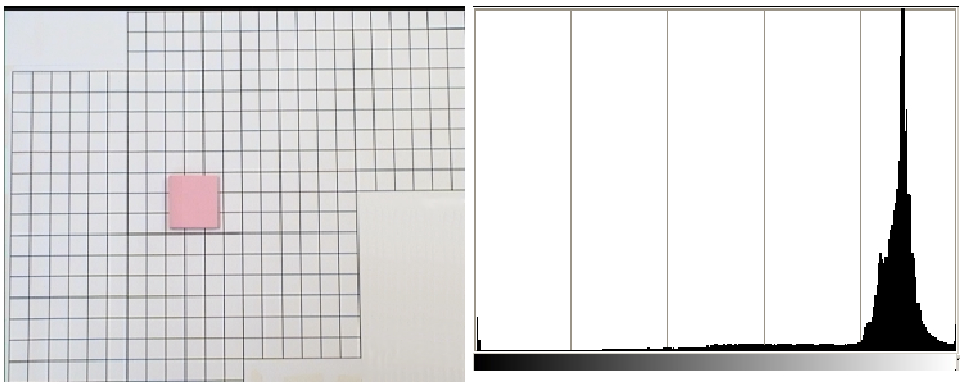
V současné době probíhá vývoj rozhraní pro jazyky jako je Python, Ruby nebo Matlab. V bakalářské práci je využita knihovna poskytující rozhraní na bázi ctypes[2], především proto, že standardní rozhraní, vytvořené pomocí wrapperu SWIG, postrádá dostatečnou dokumentaci.

4.2 Získání obrazu z kamery

Základním krokem pro zpracování obrazu je bezesporu jeho získání. Při výstavbě laboratoře byl, mimo jiné, zakoupen ovládací software ke kamerám Milestone XProtect Basis+, který dokáže integrovat obraz z několika kamer současně a samozřejmě umožňuje pohodlné ovládání kamer. Bohužel, přestože výrobce na webových stránkách zmiňuje existenci SDK (Software Development Kit), není tento balík volně ke stažení.

Jednou z možností, jak obraz z kamer získat, je při typických aplikacích pro zpracování obrazu využití knihovny OpenCV. Ta dokáže poměrně bez problémů přistupovat k nejrůznějším typům kamer, počínaje webovými kamerami integrovanými v noteboocích a kamerami s rozhraním FireWire konče. IP kamery Axis však přímo podporovány nejsou. Ačkoli jsem nenašel potvrzené informace, většina zdrojů na internetu se shoduje na tom, že problém tkví v nestandardním formátu Motion JPEGu [7] [8] [9]. Také proto jsem se rozhodl použít k propojení těchto kamer a OpenCV knihovnu, kterou jsem dříve vytvořil společně s Petrem Schindlerem. Tato knihovna je podrobněji popsána v kapitole 5.

Při zpracování obrazu je také potřeba zajistit buď neměnné osvětlení, nebo průběžně upravovat optické vlastnosti kamery (clonu) tak, aby obraz vykazoval co nejpodobnější vlastnosti. V tomto projektu využívám spíše druhý přístup. Ve zpracovávaných obrazech se v tomto případě vyskytuje nejčastěji bílá barva podlahy (viz obrázek 4.1).



Obrázek 4.1 – Běžná scéna a její histogram. Je zřetelně vidět, že většinu plochy vyplňují téměř bílé pixely.

Experimentálně jsem zjistil, že pro následné zpracování je nejlepší, pokud má histogram maximum okolo hodnoty 227. Před získáním obrazu pro detekci čar Houghovou transformací proto nejprve volám proceduru `prepareCamera()` z modulu `lib_camera.py`, která iterativně získává z

[2] http://code.google.com/p/ctypes-opencv/

kamery referenční obraz a provádí nastavení clony tak, aby se výsledek co nejvíce přiblížil zadanému histogramu.

```
def prepareCamera(
      camera,
      hist_max = 227
    )
```

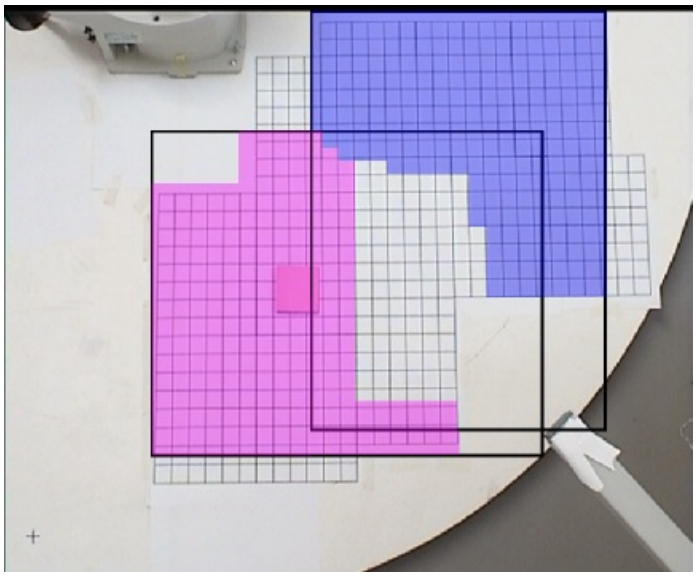
Parametr `camera` specifikuje číslo kamery, která bude kalibrována. Toto číslo lze získat například jako poslední byte IP decimálně nebo jako řetězec. Například pro IP adresu `169.254.0.11` je to číslo `11` nebo řetězec `".11"`. Parametr `hist_max` určuje maximální pozici maximální hodnoty v histogramu, ke které se má kalibrace přiblížit. Přednastavená hodnota 227 je zvolena proto, že algoritmy pro přípravu obrazu předpokládají na vstupu obraz s histogramem, jehož maximum se pohybuje právě kolem tohoto čísla.

Naskýtá se otázka, zda by nebylo vhodnější tuto funkcionalitu začlenit přímo do knihovny pro ovládání kamer. Tato knihovna je však myšlena jako minimalistický základ umožňující ovládání samotné kamery a další rozšíření považuji za vhodné řešit až v rámci projektu, který ji bude využívat.

4.2.1 Výběr kamery

Protože snímek celé pracovní plochy v laboratoři neposkytuje kvůli nízkému rozlišení obrazu kamer Axis 214 dostatečnou úroveň detailů, je pro výpočet polohy využíván obraz přiblížené části pracovní plochy. Je tedy nutné rozhodnout, která kamera poskytne lepší data pro další zpracování.

Výběr probíhá tak, že na snímku pracovní plochy jsou vymezeny oblasti (viz obrázek 4.2), ve kterých se pro každou kameru počítá počet barevných pixelů (tedy pixelů, které reprezentují obraz kostky). Následně je porovnán počet detekovaných pixelů a použije se ta kamera, pro kterou je jejich počet větší.



Obrázek 4.2 – Výběr kamery. Pixely pod fialovou plochou se počítají pro první kameru, modrá plocha pak pokrývá pixely význačné pro druhou kameru

Zřejmá nevýhoda tohoto přístupu spočívá v tom, že pokud bude na pracovní ploše více stejnobarevných kostek, stane se nespolehlivým. Při reálném nasazení se však nejspíše bude zpracovávat obraz ze všech kamer a výstupy budou vzájemně porovnány pro případnou korekci chyb.

Výběr kamery provádí funkce `selectCamera()` taktéž umístěná v knihovně `lib_camera.py`. Na výstupu generuje hodnoty `".10"` nebo `".11"`, podle toho která kamera bude pro zpracování použita.

4.3 Předzpracování obrazu

Protože podmínky v laboratoři nejsou dokonalé, je nezbytné určité předzpracování obrazu, kterým se odstraní takové jeho části, které by mohly zapříčinit neočekávané, nebo zcela chybné výsledky procesu rozpoznání pozice kostky. Funkce, které poskytují metody využívané při přípravě obrazu ke zpracování, jsou umístěny v knihovně `prepare_image.py`. Všechny následující kroky jsou zabaleny ve funkci `prepareZoomedImageForHough()`, která je ve správném pořadí provede.

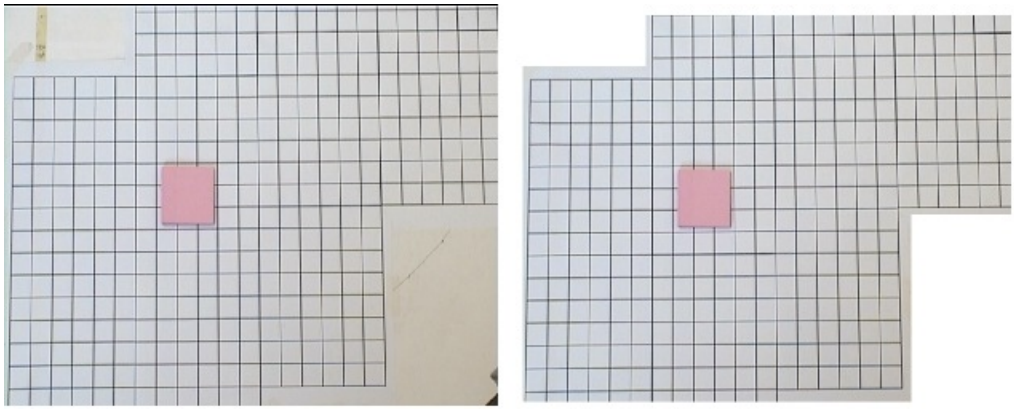
4.3.1 Ořezání obrazu

První krok spočívá v odstranění (resp. zamaskování) nežádoucích částí (viz obrázek 4.3). Většinou se jedná o viditelnou hranici pracovní plochy, ale může to být také základna robotického ramene nebo značky na pracovní ploše, které jsou pozůstatkem po předchozích experimentech s různými způsoby zaměřování. Tyto korekce provádí funkce `trimImage()`

```
def trimImage(
      image,
      camera,
      zoomed = True,
      out = "OpenCV"
    )
```

První parametr specifikuje vstupní obraz (cesta k souboru, obrázek v interním formátu knihovny Python Imaging Library[3], obrázek ve formátu OpenCV IPL), druhý parametr specifikuje číslo kamery, ze které byl obraz pořízen (poslední byte IP adresy, tedy pro adresu `169.254.0.1` číslo `10`, případně řetězec `".10"`). Třetí parametr určuje, zda se právě zpracovává pohled na celou pracovní plochu, nebo obraz přiblížený jen na její část. Poslední parametr udává typ výstupního obrazu, akceptované hodnoty jsou řetězce `"PIL"` a `"OpenCV"`.

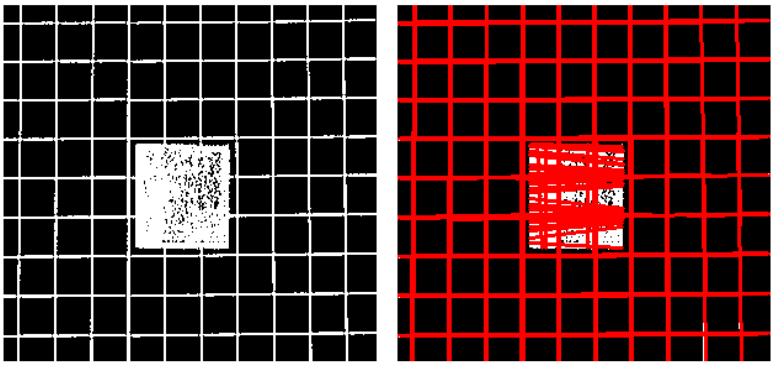
[3] http://www.pythonware.com/products/pil/



Obrázek 4.3 – Odstranění nepotřebných částí z obrázku.

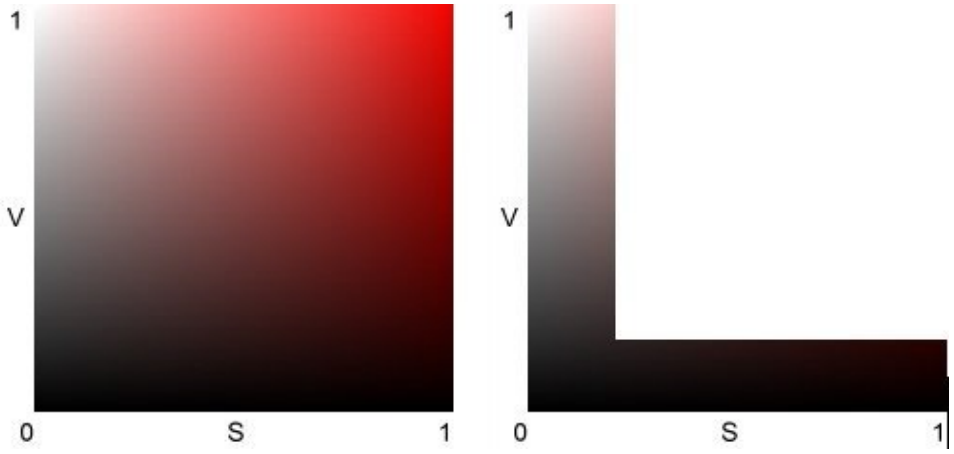
4.3.2 Odstranění barevných pixelů

Protože dalším krokem je extrakce mřížky pomocí thresholdingu, je nutné z obrazu odstranit barevné kostky. Hlavním důvodem je fakt, že při prahování záleží pouze na intenzitě barvy, a ne na jejím odstínu. Například tmavě růžová kostka by byla prahováním rozpoznána jako součást mřížky (viz obrázek 4.4), čímž by se takovýto obraz stal nepoužitelným pro Houghovu transformaci.



Obrázek 4.4 – Výsledek detekce čar Houghovou transformací bez předchozího odstranění kostky.

Proto je nutné barevné pixely detekovat a odstranit. Pokud převedeme problém detekce barevných pixelů na nalezení doplňku k šedým pixelům, bylo by možné použít i standardní reprezentaci RGB (odstíny šedi mají všechny tři složky stejně velké). Mnohem výhodnější však je využít model HSV. Pokud se podíváme na průřez barevným prostorem (obrázek 4.5) je zřetelné, že oblast šedých pixelů lze vymezit velmi jednoduše.



Obrázek 4.5 – Průřez barevným prostorem HSV. Šedé pixely lze vymezit například jako oblast, kde je složka V nebo S menší než 0,15.[4]

Odstranění barevných pixelů provádí funkce `deleteColoredPixels()`.

```
def deleteColoredPixels(
      image,
      out = "PIL",
      minS = 0.15,
      minV = 0.15
    )
```

První parametr specifikuje vstupní obraz, druhý pak požadovaný výstupní formát obrázku. Parametry `minS` a `minV` určují spodní mez sytosti a světlosti pro barevné pixely. Tzn. pixel, jehož sytost nebo světlost přesáhnout tuto hodnotu, je detekován jako barevný a odstraněn.

Protože vstupní data reprezentují barvy ve formátu RGB, je nutné nejprve provést konverzi. Model HSV je odvozen od RGB, a proto je možné mezi nimi bezztrátově převádět. Funkce `DeleteColoredPixels()` však neobsahuje vlastní implementaci konverzního algoritmu, ale využívá funkci `rgb_to_hsv()` ze standardní knihovny Pythonu `colorsys`.

4.3.3 Extrakce mřížky užitím prahování

Prahování zajišťuje funkce `doThresholding()`, která pouze obaluje volání metody `cvThreshold()` a samozřejmě provádí všechny nezbytné konverze.

```
def doThresholding(
      image,
      out = "OpenCV",
      threshold = 190
    )
```

Parametry `image` a `out` opět udávají vstupní data a formát obrazu na výstupu. Parametr `threshold` je pak předán funkci `cvThreshold()` jako hodnota prahu.

[4] Obrázek převzat z http://www.gamutvision.com/docs/gamutvision_equations.html

4.3.4 Perspektivní transformace

Přestože Houghova transformace by měla být nezávislá na vzájemných úhlech čar v obraze, experimenty potvrdily, že lepší výsledky podává, pokud je obraz rektifikovaný, tedy mřížka pravoúhlá. Funkce `doPerspectiveTransform()` opět zapouzdřuje volání metod z OpenCV.

Pro každou kameru jsou předdefinovány transformační souřadnice – souřadnice čtyř bodů v původním obraze a příslušné souřadnice v obraze transformovaném. Z nich metoda `cvGetPerspectiveTransform()` vypočte transformační matici, která je pak použita při volání funkce `cvWarpPerspective()`.

```
def doPerspectiveTransform(
      image,
      camera,
      out = "OpenCV"
    )
```

Význam parametrů je již dostatečně vysvětlen v předchozích sekcích. Je snad nutné zmínit jen to, že současná implementace nepředpokládá nutnost provádění transformace nad záběry celého pracovního prostoru, ale počítá s tím, že na vstupu je obraz již přiblížené části. Proto na rozdíl od funkce `trimImage()` nemá parametr `zoomed`.

4.4 Rozpoznání pozice kostky

Poté, co je obraz z kamery připraven, je možné přikročit k samotné detekci kostky. Z obrazu je nejprve pomocí Houghovy transformace extrahována mřížka. V ní jsou následně nalezeny „díry“, jejichž pozice je dále analyzována a přepočítána na reálné souřadnice kostky na pracovní ploše. Hlavním rozhraním pro získání dat o pozici kostky je funkce `getCubes()` v souboru `detection.py`. Tato funkce postupně provádí všechny kroky potřebné k zaměření pozice kostky. Jejím výstupem je množina středů zaměřených kostek.

4.4.1 Detekce mřížky

Detekce mřížky je implementována v modulu `grid_detect.py`, funkcí `detectGrid()`. Zde se přiblížený obraz pracovní plochy nejprve předpřipraví. Následně zavolá detekci čar Houghovou transformací a sloučí fragmentované úseky. Výstupem je seřazený seznam vodorovných a svislých částí mřížky.

4.4.1.1 Houghova transformace

Houghova lineární transformace je typickým nástrojem pro detekci čar v obraze. Knihovna OpenCV poskytuje dvě hlavní implementace. Standardní Houghovu transformaci (SHT) a progresivní pravděpodobnostní Houghovu transformaci (PPHT). Rozdíly ve výsledcích SHT a PPHT jsou demonstrovány na obrázku 3.6 v kapitole 3.1.3.

Pro výběr PPHT nebyl důležitý jen fakt, že detekuje také délku čar, ale především to, že podává lepší výsledky v případě, že jsou na obraze různě dlouhé čáry. Pro kvalitu detekce blížící se výsledkům PPHT pomocí SHT je nutné nastavit velmi malý práh. SHT pak sice detekuje většinu čar, ale generuje tisíce objektů, které musí být zvlášť porovnány. PPHT sice na testovacích datech

rozpoznává čáry fragmentovaně, ale i při velmi nízkém prahu (u PPHT je to kombinace tří parametrů), je počet rozpoznaných objektů maximálně v řádu stovek.

Použití implementace PPHT z knihovny OpenCV (funkce `cvHoughLines2()`) je zapouzdřeno ve funkci `doHoughTransform()` v modulu `lib_hough.py`.

```
def doHoughTransform(
      image,
      threshold,
      param1,
      param2
    )
```

První parametr udává vstupní obraz – předpokládá se, že této metodě je předán již předzpracovaný binární obraz. Parametry `threshold`, `param1` a `param2` jsou přímo předány funkci `cvHoughLines2()` dle [2] je jejich význam následující: `threshold` je hodnota, která musí být dosažena v akumulátoru, aby byla detekována čára. `Param1` udává minimální délku segmentu čáry potřebnou k tomu, aby byl tento segment vrácen. `Param2` nastavuje minimální vzdálenost mezi kolineárními segmenty nutnou k tomu, aby nebyly spojeny do jednoho delšího segmentu.

Výstupem funkce `doHoughTransform()` jsou dvě pole – první obsahuje vodorovné a druhé svislé segmenty čar.

4.4.1.2 Sloučení segmentů čar

Protože PPHT generuje pro každou čáru několik segmentů, je vhodné je před další analýzou sloučit. Tato „defragmentace“ je implementována v metodě `_oneFromMany()` v souboru `grid_detect.py`.

```
def _oneFromMany(
      from_hough,
      orientation
    )
```

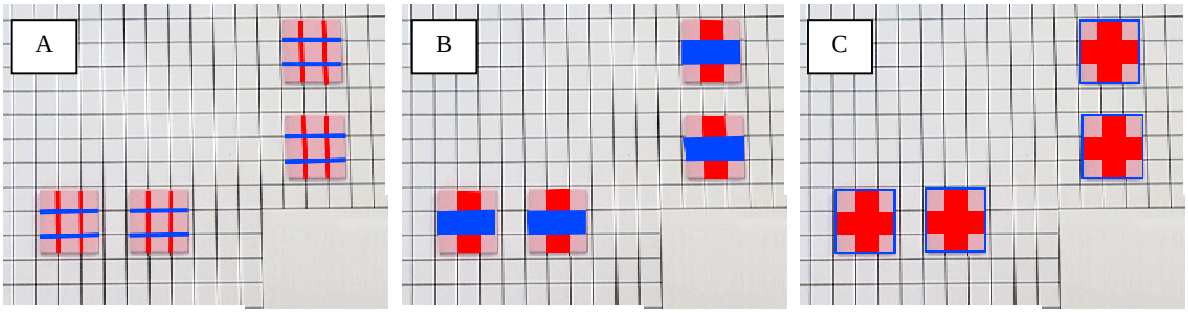
Parametr `fromHough` je pole fragmentů, `orientation` udává, zda se jedná o vodorovné (hodnota `"horizontal"`) nebo svislé (hodnota `"vertical"`) segmenty. Výstupem funkce je pole jednotlivých čar mřížky. Postup sloučení úseků do jedné úsečky demonstruje následující schéma:

1. Rozdělení fragmentů do řádků, podle pozice na ose y
2. Seřazení fragmentů v řádku podle souřadnice počátečního bodu na ose x.
3. Sjednocení překrývajících se částí do spojité čáry, respektive čar, pokud je v mřížce mezera.
   Pro každý řádek:
   a. Inicializace začátku a konce čáry na souřadnice počátečního bodu prvního segmentu.
   b. Přesun prvního segmentu na řádku do pomocné proměnné.
   c. Pokud leží počáteční bod segmentu uvnitř již zpracované čáry nebo v rozmezí 10px od jejího konce, pak:
      – Pokud je koncový bod segmentu více vpravo, než aktuální konec čáry, nastaví se koncová souřadnice čáry na koncovou souřadnici segmentu.
      – Návrat na řádek b.
   d. Jinak – uložení zpracované čáry do seznamu čar a pokračování na řádku a

4.4.2 Detekce kostky

Výstupem funkce `detectGrid()`(viz kap.4.4.1) je seznam jednotlivých čar v mřížce, který následně zpracovávají metody v knihovně `cube_detect.py`. Pro detekci pozice kostky je však důležité mít informace o tom, kde je mřížka přerušena na základě funkce `_getGaps()`, jíž je na vstup předán seznam čar z `detectGrid()`.

Dále je nutné zjistit, která přerušení mřížky se „protínají“ a vytyčují tak plochu zabranou kostkou. Tento proces je rozdělen do dvou kroků. Nejprve jsou vytvořeny obdélníky zvlášť pro sousedící vodorovné a svislé mezery pomocí funkce `_squaresFromGaps()`. Vytvořené obdélníky jsou pak vzájemně testovány funkcí `_intersectSquares()` a u těch, které se protínají, je vytvořen minimální ohraničující obdélník, který je prohlášen za reprezentaci polohy kostky v mřížce (viz obrázek 4.6).



Obrázek 4.6 – Detekce kostky **(A)** Rozpoznané mezery ve mřížce, **(B)** převedené na obdélníky. **(C)** Ohraničující čtverec kolem protínajících se obdélníků je považován za detekovanou kostku

Výstupem funkce `_intersectSquares()` je pole objektů typu `Square`, který je definován v modulu `common.py`. Každý objekt třídy `Square` obsahuje jak informace o poloze obdélníku na obrázku v pixelech, tak údaje o tom, které linie mřížky jej ohraničují. Tyto informace jsou pak využity pro výpočet skutečné polohy kostky. Volání všech zmíněných funkcí ve správném pořadí zajišťuje metoda `detectCubesInImage()`.

```
def detectCubesInImage(
      image,
      camera
    )
```

Parametr `image` určuje obrázek, který se bude zpracovávat (cesta k souboru, obrázek ve formátu PIL nebo obrázek ve formátu OpenCV IPL), `camera` pak definuje, kterou kamerou byl pořízen.

4.4.3 Výpočet pozice kostky na pracovní ploše

Pro výpočet skutečných souřadnic kostky lze použít dva přístupy. Jedním je určení pozice na základě nejbližších viditelných čar v mřížce kolem detekovaného obdélníku. Pokud známe skutečnou pozici nejhornější a nejlevější linky, a také rozteč mezi jednotlivými čarami mřížky, lze přibližnou pozici určit dopočítáním vzdálenosti na základě počtu linek shora a zleva od rozpoznaného obdélníka.

Druhý způsob využívá pro převod souřadnice obdélníku na obrázku v pixelech. Pro rektifikovaný obraz určíme skutečnou polohu levého horního rohu a rozměry pixelu v milimetrech. Šířka a délka pixelu se po převedení do reálného souřadného systému liší, přibližně však svými rozměry pixel odpovídá jednomu milimetru. Kombinací těchto hodnot a známé polohy jednoho bodu v obraze lze dopočítat souřadnice levého horního rohu obrazu a následně dopočítat polohu kostky.

Výpočet provádí funkce `_calculateCubePosition()`.

```
def _calculateCubePosition (
      square,
      camera,
      cal_in_img
    )
```

Parametr `square` je instance třídy `Square`, která obsahuje informace o pozici levého horního a pravého dolního bodu detekovaného obdélníka na obrázku a také čísla čar, které tento obdélník přerušuje (výstup funkce `detectCubesInImage()`). Parametr `camera` určuje, kterou kamerou byl pořízen obraz pro detekci. Tento parametr je důležitý především pro výběr správné délky a šířky pixelu. Poslední parametr `cal_in_img` obsahuje pozici kalibračního bodu pro danou kameru na získaném obraze. Tuto hodnotu taktéž generuje funkce `detectCubesInImage()`.

Celý proces detekce pozice kostky zapouzdřuje metoda `detectCubes()`, taktéž z knihovny `cube_detection.py`.

4.5 Testování a výsledky

Pro testování algoritmu jsem vytvořil 51 testovacích trojic snímků, na kterých kostka postupně pokryje většinu pracovní plochy. Každá trojice sestává z oddáleného snímku pracovní plochy (především pro pozdější referenci, případně také k testování algoritmu pro výběr kamery) a přiblížených snímků z obou kamer. Tato testovací data jsou přiložena na DVD. Ke každé testovací množině je také připojen popisný soubor meta_in.py, který obsahuje informace o předpokládaném výběru kamery a o pozici středu kostky v souřadném systému robotického ramene. Střed kostky jsem jako hodnotu pro testování zvolil proto, že při rozpoznávání pozice dochází k chybě způsobené rozdílnou šířkou pixelů v reálném prostoru a zprůměrováním pozic levého horního a pravého dolního rohu kostky je tato chyba mírně potlačena. Lepším řešením by samozřejmě bylo vytvořit bitmapu reprezentující skutečné rozměry každého pixelu a vzdálenosti měřit v ní.

Pro každý testovací soubor jsem spustil algoritmus rozpoznávající polohu kostky a výsledná data jsem uložil, společně s vizualizací rozpoznané mřížky a kostek. Tyto výsledky jsou také přiloženy na DVD. Výsledky testů jsou poměrně rozporuplné. Hodnoty zaměřených pozic odpovídají vcelku přesně předpokládaným hodnotám, ale při zpracování obrazu z první kamery, dochází velmi často k problémům při detekci mřížky Houghovou transformací. Po analýze dat, které generuje Houghova transformace, předpokládám, že problém tkví ve stínech, které kostka vrhá na pracovní plochu. Ty jsou, pravděpodobně vlivem mírně tmavších snímků z této kamery, rozpoznány jako stovky velmi krátkých úseků čar a pravděpodobně způsobí problém funkci, která provádí jejich seskupování. Řešení by mohlo poskytnout předběžné filtrování detekovaných úseků podle prostorových vlastností (např. detekce velkého množství krátkých vodorovných úseků s minimálními vertikálními vzdálenostmi), případně použití jiného algoritmu pro detekci čar. Pokud se ale budeme zabývat pouze přesností rozpoznaných pozic, poskytuje tato metoda dostatečně kvalitní výsledky, jak

při rozpoznání pozice pomocí nejbližších viditelných ohraničujících linií mřížky, tak při detekci pozice přepočítáním souřadnic kostky na obraze do reálného prostoru.

5 Ovládání kamer Axis 214

Samozřejmou nutností při rozpoznávání obrazu, je ovládáni kamer. Laboratoř je vybavena IP kamerami Axis 214, k nimž byl zakoupen taktéž ovládací software umožňující pohodlné ovládání kamery a poskytující nejrůznější funkce vhodné především pro dohledová centra atp. Ačkoli výrobce tohoto softwaru údajně poskytuje SDK (Software Development Kit), pomocí kterého by mělo být možné využít tento program jako backend pro ovládání kamer a získávání obrazu v uživatelských aplikacích, není tento balík knihoven na webu výrobce ke stažení, a zdá se, že je poskytován pouze za úplatu.

Proto jsem společně s Petrem Schindlerem vytvořil knihovnu pro jazyk Python, pomocí které lze kamery Axis 214 (a další IP kamery tohoto výrobce) ovládat. Prvním krokem při získávání informací o způsobu ovládání byla analýza webového rozhraní kamer. Při zkoumání zdrojových textů ovládacích skriptů jsme narazili na zmínku o protokolu VAPIX [1], který je, jak jsme zjistili, standardizovaným API pro ovládání IP kamer společnosti AXIS. Námi vytvořená knihovna tedy zasílá kamerám HTTP požadavky splňující standard VAPIX protokolu a případně zpracuje příchozí data z kamery.

5.1 Popis knihovny

Knihovna je umístěna v souboru `kamera.py` a poskytuje metody jak pro pohyb s kamerou, tak pro nastavení clony a zoomu. Přestože kamery Axis 214 disponují řadou dalších možností (např. noční vidění, detekce pohybu atd.), tyto možnosti nejsou v knihovně implementovány, protože pro potřeby práce v laboratoři nejsou důležité.

Veškerá funkcionalita knihovny je obsažena ve třídě `Camera`. Při vytvoření nové instance této třídy je nutné konstruktoru předat IP adresu kamery, login a heslo k webovému rozhraní, jehož služeb knihovna využívá. Třída `Camera` pak poskytuje rozhraní pro manipulaci s kamerou. Ačkoli lze pro ovládání kamery použít metody této třídy, jako jsou např. `set_pan()` pro přímé nastavení hodnoty vodorovného natočení nebo `rel_pan()` pro relativní pohyb s kamerou[5], je výhodnější využít tzv. properties – tedy vlastností objektu, které jistým způsobem nastavování hodnot zapouzdřují:

```
from kamera import Camera

c = Camera(ip_adresa, "login", "heslo")    #inicializace
print c.pan    #vypíše aktuální horizontální natočení kamery
c.pan = 90     #nastaví horizontální otočení na 90
c.pan -= 10    #otočí kamerou o 10 stupňů vlevo od současné pozice
```

Tyto properties jsou `pan` pro horizontální natočení, `tilt` pro vertikální natočení, `iris` pro nastavení clony a `zoom` pro přiblížení/oddálení. Pro získání obrazu z kamery pak slouží především properties `bmp_PIL_frame` a `jpg_PIL_frame`, které poskytují aktuální záběr z kamery v reprezentaci používané knihovnou Python Imaging Library.

[5] `set_pan(90)` otočí kameru do pozice, kdy směřuje přesně vpravo. `rel_pan(-30)` otočí kamerou o 30° vlevo oproti současnému natočení

6 Závěr

V rámci bakalářské práce se podařilo splnit všechny body zadání, tedy navrhnout a implementovat metodu rozpoznávání pozice kostky na pracovní ploše robotického ramene Mitsubishi Melfa 6 SL.

Zvolená metoda založená na detekci čtvercové mřížky zakreslené na pracovní ploše a následné analýze přerušení v mřížce vytvořených položenou kostkou poskytuje vyhovující výsledky, co se týká přesnosti zaměřených souřadnic. Naráží však na problémy při detekci samotné mřížky. Houghova transformace, která je pro detekci čar používána, často generuje velké množství falešných segmentů čar, případně má problémy s rozpoznáním krátkých linií – např. pokud je kostka položena blízko kraje obrazu.

Dalším krokem ve vývoji tohoto projektu by tedy mohla být buďto implementace jiného algoritmu pro rozpoznávání čar, nebo, což je dle mého názoru vhodnější varianta, přechod na rozpoznávání pozice pomocí přímého přepočtu z rektifikovaného obrazu. Mřížka by pak nebyla použita přímo pro výpočet pozice, ale pouze jako podklad pro přesnou rektifikaci získaného obrazu a případně pro korekci vypočtených souřadnic. Tímto problémem se budu zabývat v další práci na této části projektu online laboratoře.

Jako vedlejší produkt této bakalářské práce vznikla knihovna pro ovládání IP kamer firmy Axis, která poskytuje jednoduché intuitivní rozhraní a lze ji využít i v jiných projektech. Například při vytvoření online pracovního místa v laboratoři L306, o jehož realizaci se v současné době uvažuje.

Literatura

[1] Axis Communications: *VAPIX, HTTP API Specification.* [online].
URL <http://www.axis.com/techsup/cam_servers/dev/cam_http_api_2.php>

[2] Bradski, G.; Kaehler, A.: *Learning OpenCV.* O'Reilly Media, 2008, ISBN 978-0-596-51613-0.

[3] Davies, E. R.: *Machine Vision: Theory, Algorithms, Practicalities.* Morgan Kaufmann, třetí vydání, 2005, ISBN 0-12-206093-8.

[4] Fisher, R.; Perkins, S.; Walker, A.; aj.: *Hough Transform.* [online]. [cit. 2009-05-04].
URL <http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>

[5] Fisher, R.; Perkins, S.; Walker, A.; aj.: *Thresholding.* [online]. [cit. 2009-05-04].
URL <http://homepages.inf.ed.ac.uk/rbf/HIPR2/threshld.htm>

[6] Fixtars Corporation: *CVCell - Module developed by Fixstars that accelerates OpenCV Library for the Cell/B.E. processor.* [online]. [cit. 2009-05-15].
URL <http://www.fixstars.com/en/company/press/20071128.html>

[7] Google Groups: *How to import video from IP Camera.* [online]. [cit. 2009-05-10].
URL <http://tinyurl.com/axis214-opencv-2>

[8] Google Groups: *IP camera with OpenCV using cvCreateFileCapture.* [online].[cit. 2009-05-10].
URL <http://tinyurl.com/axis214-opencv-1>

[9] Kodubets, A.: *Getting MJPEG stream from Axis IP-Camera*. [online]. [cit. 2009-05-10].
URL <http://tinyurl.com/axis214-opencv-3>

[10] Kršek, P.: *Základy počítačové grafiky - studijní opora.* [verze 0.9].

[11] Matas, J.; Galambos, C.; Kittler, J.: *Progressive Probabilistic Hough Transform.* In British Machine Vision Conference, 1998.

[12] Python Software Foundation: *About Python*. [online]. [cit. 2009-05-15].
URL <http://www.python.org/about/>

[13] Schindler, P.: *Ovládání robotického ramena Mitshubishi Melfa 6 SL*. Bakalářská práce, Fakulta informačních technologií Vysokého učení technického v Brně, 2009.

[14] Stančík, P.: *Principle of Stereophotogrammetry - 3D Point Coordinates Reconstruction*.
In Sborník prací konference a soutěže Student EEICT 2004, 2004.

[15] Wikipedia: *Color model*. [online]. [cit. 2009-05-03].
URL <http://en.wikipedia.org/wiki/Color_model>

[16] Wikipedia: *Hough transform*. [online]. [cit. 2009-05-04].
URL <http://en.wikipedia.org/wiki/Hough_transform>

[17] Wikipedia: *HSV color space*. [online]. [cit. 2009-05-03]
URL <http://en.wikipedia.org/wiki/HSV_color_space>

[18] Wikipedia: *Image histogram.* [online]. [cit. 2009-05-04].
URL <http://en.wikipedia.org/wiki/Image_histogram>

[19] Wikipedia: *OpenCV*. [online]. [cit. 2009-05-15].
URL <http://en.wikipedia.org/wiki/OpenCV>

[20] Wikipedia: *RGB color model*. [online]. [cit. 2009-05-03].
URL <http://en.wikipedia.org/wiki/RGB_color_model>

[21] Wikipedia: *Triangulation.* [online]. [cit. 2009-05-04].
URL <http://en.wikipedia.org/wiki/Triangulation>

[22] Zisserman, A.; Hartley, R.: *Multiple View Geometry in Computer Vision.* Cambridge University Press, druhé vydání, 2004, ISBN-13 978-0521540513.

Seznam příloh

Příloha 1 – Diagram volání funkcí